

**Program :**B.tech(CSE)

**Specialization :**AIML

**Course Title :**AI Assisted Coding

**Course Code :**24CS002PC215

**Semester :**3rd semester

**Academic Session :**2025-2026

**Name of Student :**Kaveti Manohar

**Enrollment No. :**2403A52079

**Batch No. :**02

**Date :**07/10/2025

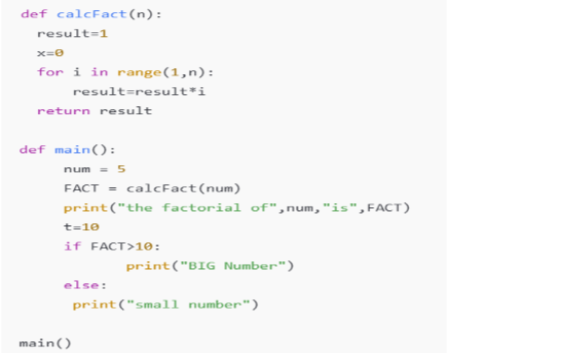
#LAB ASSIGNMENT

Task Description -1:

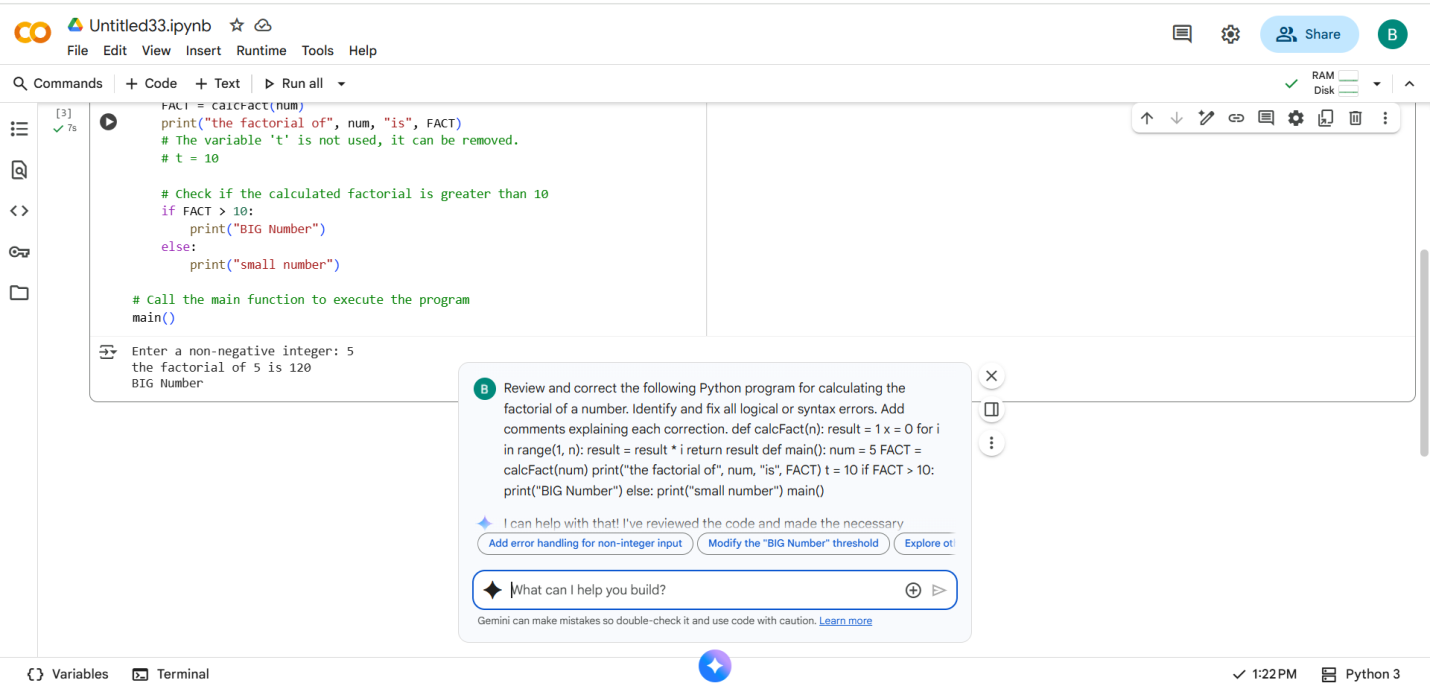
Write python program as shown below.  
 Use an AI assistant to review and suggest corrections..

PROMPT:

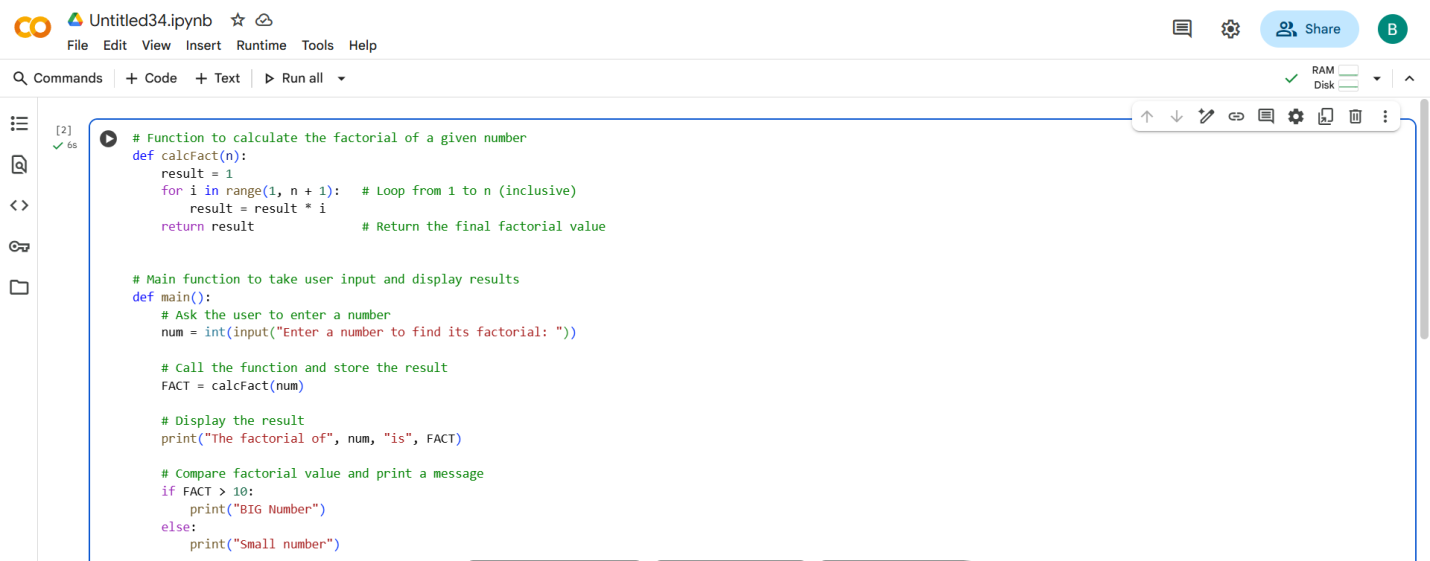
* Review and correct the following Python program for calculating the factorial of a number. Identify and fix all logical or syntax errors. Add comments explaining each correction.

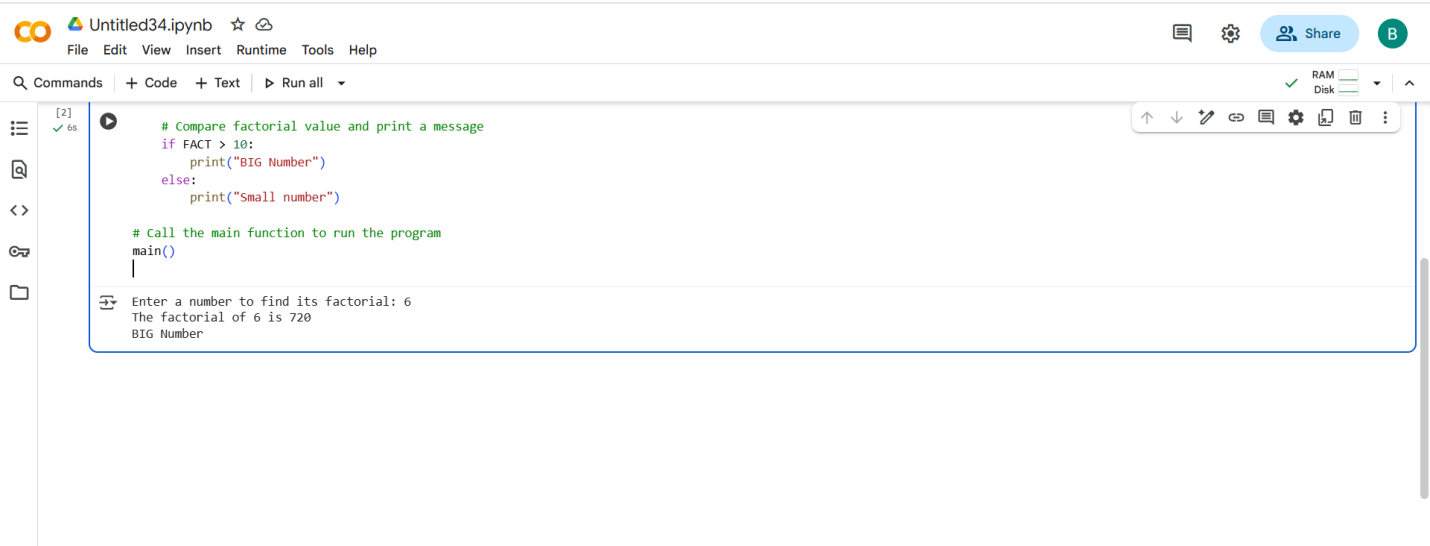


Question:

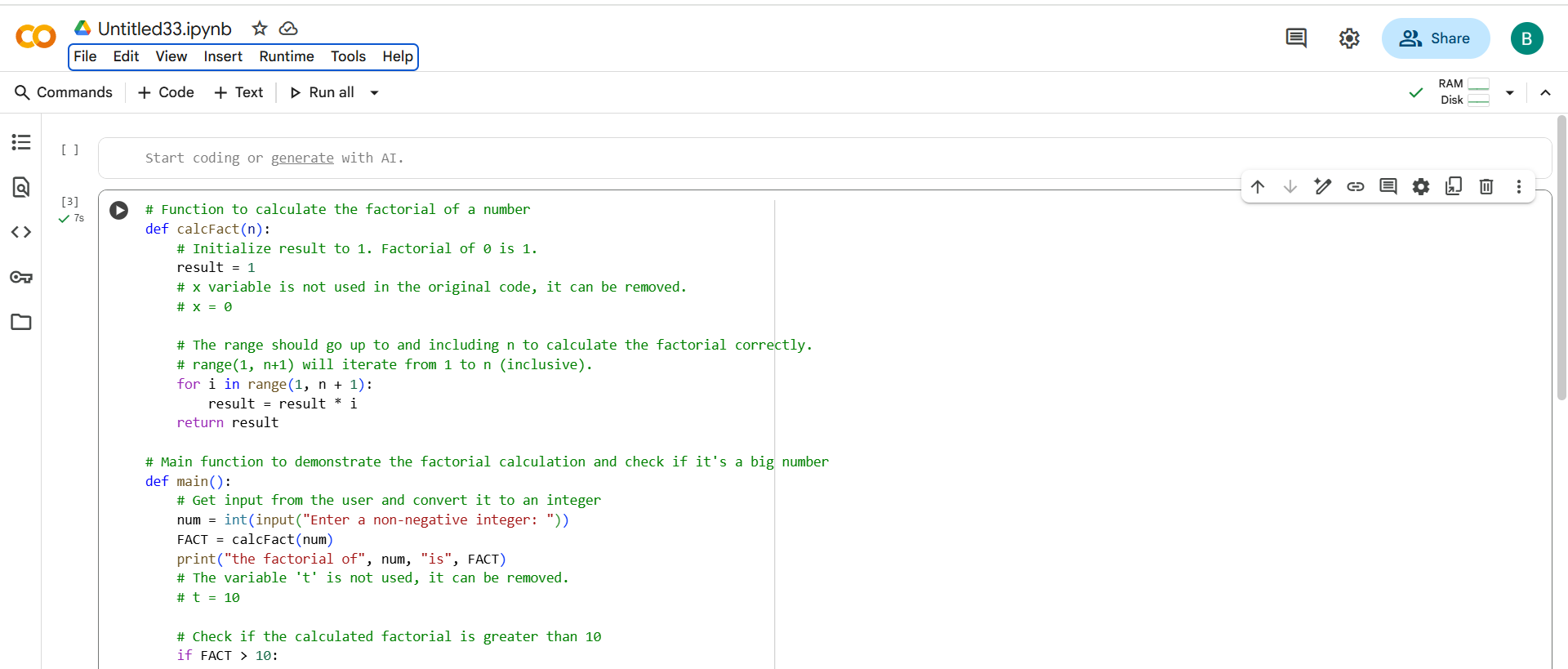


Code with output corrected by manual:





Code with output corrected with AI:





* Comparing both codes

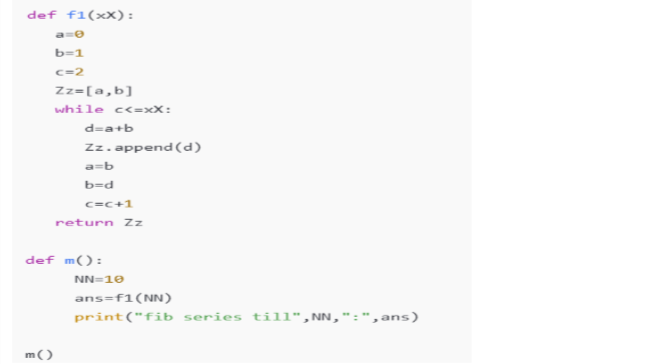
By comparing both codes Ai written well but which code I written it is more understandable compare to AI.

EXPLANATION:

* generate\_fibonacci\_series(num\_terms): This function takes the desired number of terms as input. It initializes a list with the first two Fibonacci numbers (0 and 1). It then iteratively calculates the next number in the series by adding the previous two, appending each new number to the list until the desired number of terms is reached.
* main(): This function prompts the user to enter the number of terms, calls generate\_fibonacci\_series() to get the series, and then prints the resulting Fibonacci series.

TASK Description-2:

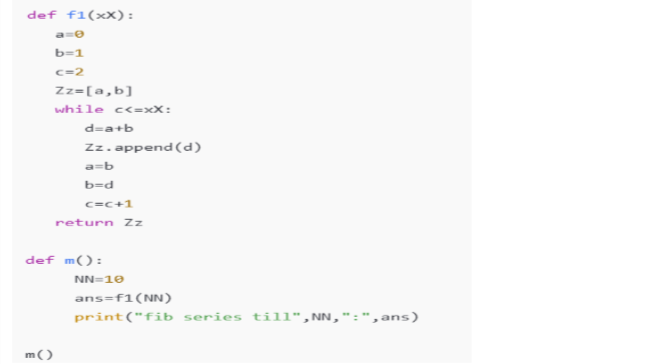
Write the Python code for Fibonacci as shown below and execute.  
• Ask AI to improve variable names, add comments, and apply PEP8 formatting  
(cleaned up).  
• Students evaluate which suggestions improve readability most. one.  
Expected Output#2: Clean format python code with much readability.



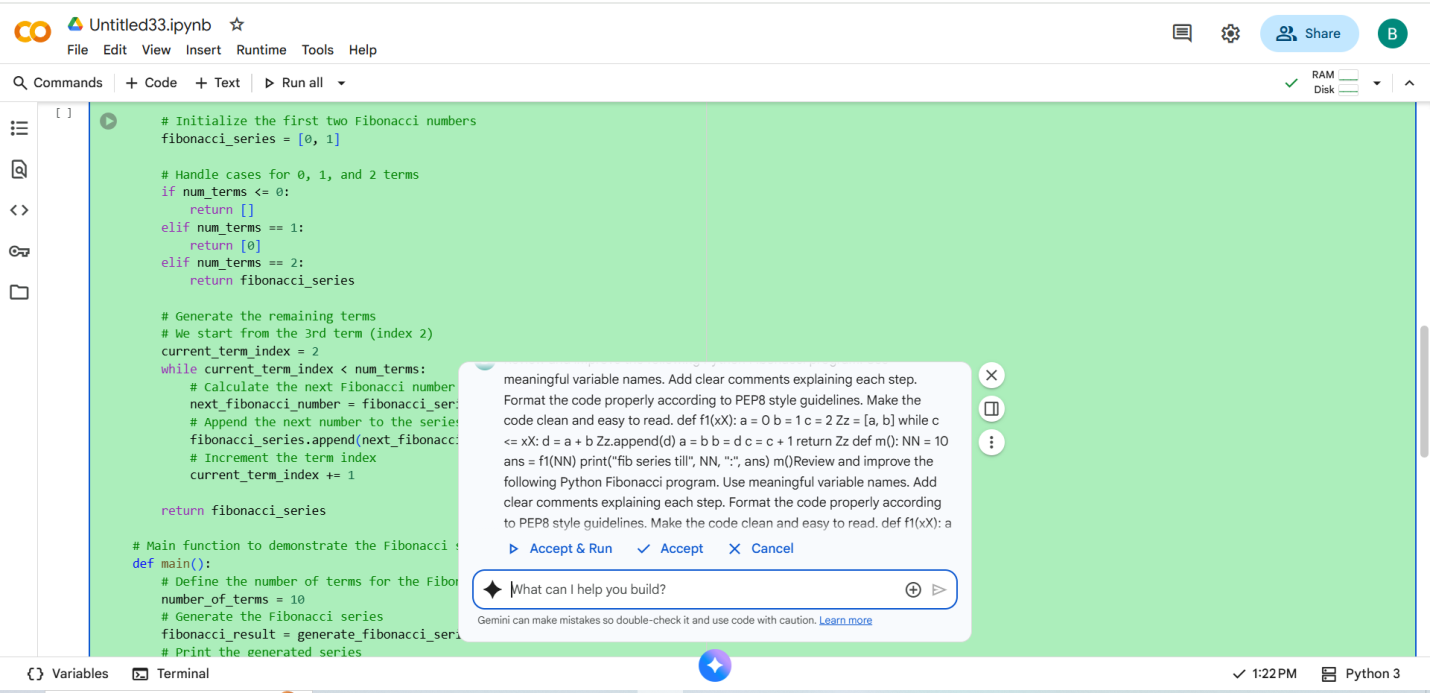
PROMPT:

Review and improve the following Python Fibonacci program.

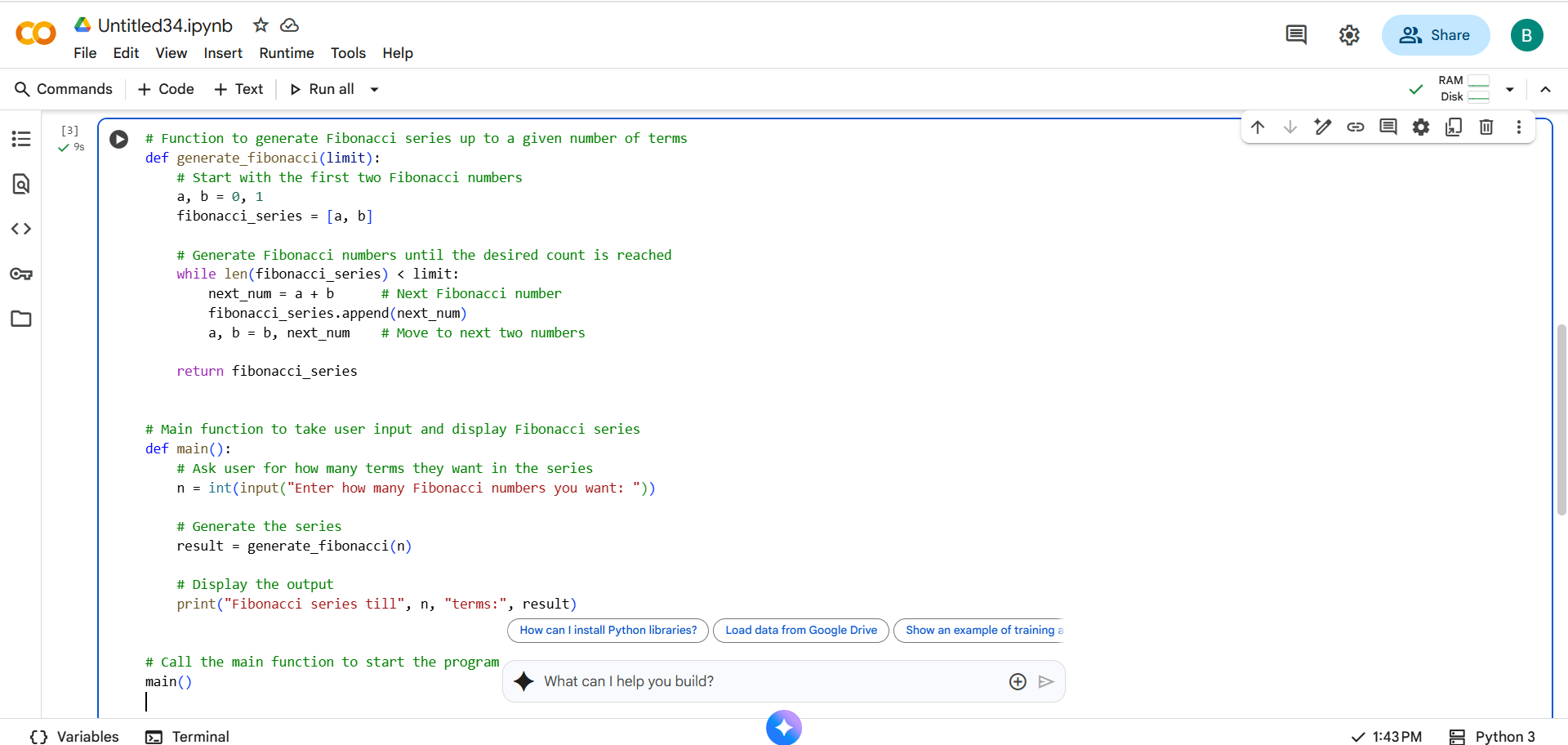
* Use meaningful variable names.
* Add clear comments explaining each step.
* Format the code properly according to PEP8 style guidelines.
* Make the code clean and easy to read.

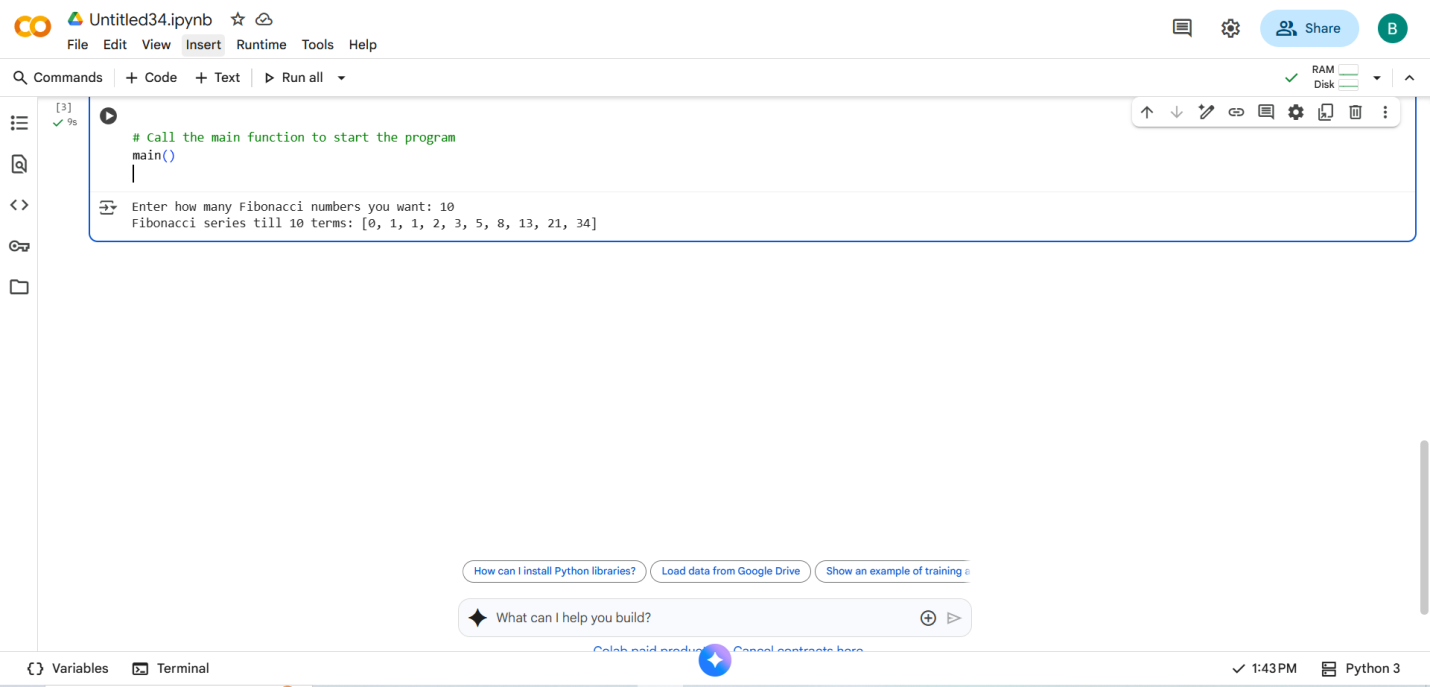


Question:

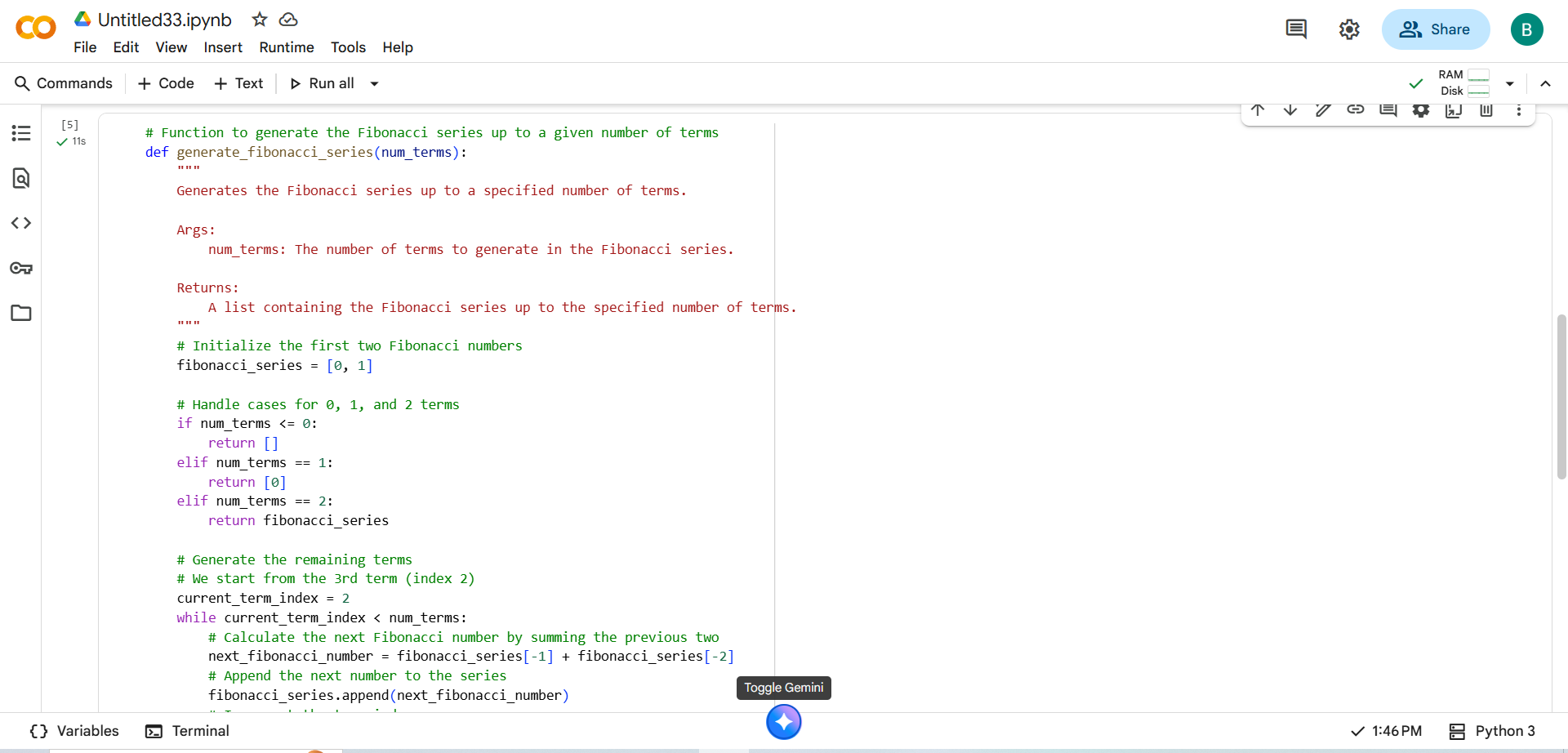


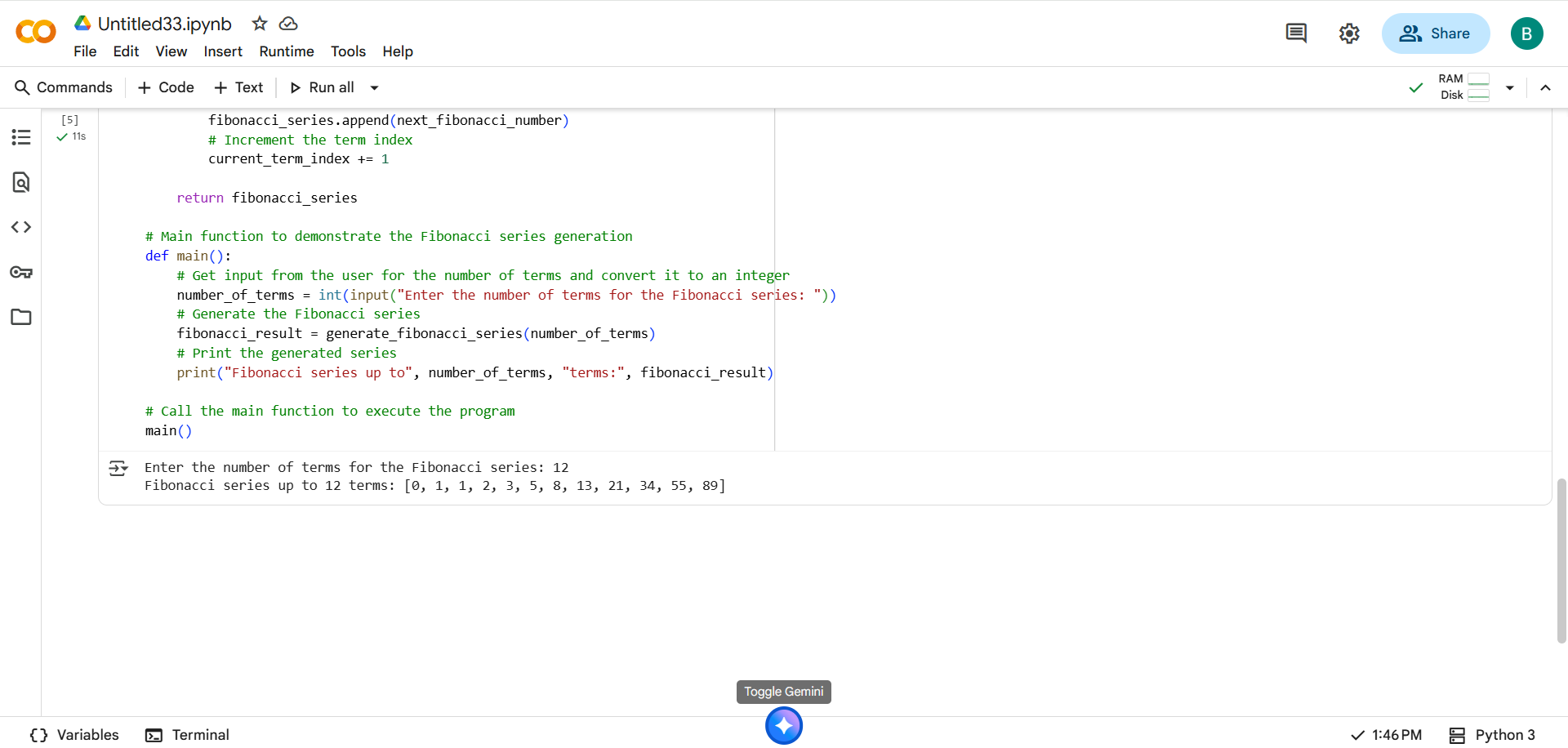
* Code with output corrected by manual:





* Code with output corrected with AI:





* Compare both codes :
* By comparing both codes almost both codes are similar code and similar output I got , but AI gives the diffent ways for output.

EXPLANATION:

* generate\_fibonacci\_series(num\_terms): This function is designed to produce the Fibonacci sequence. It takes one argument, num\_terms, which is the desired length of the series. It starts by creating a list fibonacci\_series with the initial two numbers of the sequence, 0 and 1. It includes checks for edge cases where num\_terms is 0, 1, or 2. If num\_terms is greater than 2, it enters a while loop that continues until the list contains the specified number of terms. Inside the loop, it calculates the next Fibonacci number by adding the last two numbers in the list and then appends this new number to the list.
* main(): This function is the main execution part of the script. It prompts the user to enter the number of terms for the Fibonacci series using the input() function and converts this input to an integer. It then calls the generate\_fibonacci\_series() function with the user's input to get the Fibonacci sequence. Finally, it prints the generated Fibonacci series to the console.

TASK Desciption-3:

Write a Python script with 3–4 functions (e.g., calculator: add, subtract, multiply,  
divide).  
• Incorporate manual docstring in code with NumPy Style  
• Use AI assistance to generate a module-level docstring + individual function  
docstrings.  
• Compare the AI-generated docstring with your manually written one.  
Common Examples of Code Smells  
• Long Function – A single function tries to do too many things.  
• Duplicate Code – Copy-pasted logic in multiple places.  
• Poor Naming – Variables or functions with confusing names (x1, foo, data123).  
• Unused Variables – Declaring variables but never using them.  
• Magic Numbers – Using unexplained constants (3.14159 instead of PI).  
• Deep Nesting – Too many if/else levels, making code hard to read.  
• Large Class – A single class handling too many responsibilities.  
Why Detecting Code Smells is Important  
• Makes code easier to read and maintain.  
• Reduces chance of bugs in future updates.  
• Helps in refactoring (improving structure without changing behavior).  
• Encourages clean coding practices  
Dead Code – Code that is never executed.  
Expected Output#3: Students learn structured documentation for multi-function scripts

PROMPT:

Write a Python program that acts as a simple calculator with 3–4 functions (add, subtract, multiply, divide).

* Include **manual NumPy-style docstrings** for the module and each function.
* Ensure each function has clear parameter and return type descriptions.
* Use **AI assistance** to generate docstrings and compare them with your own manual ones.
* Make the code clean, readable, and properly formatted (PEP8 style).

Example functions to include:

def add(a, b):

pass

def subtract(a, b):

pass

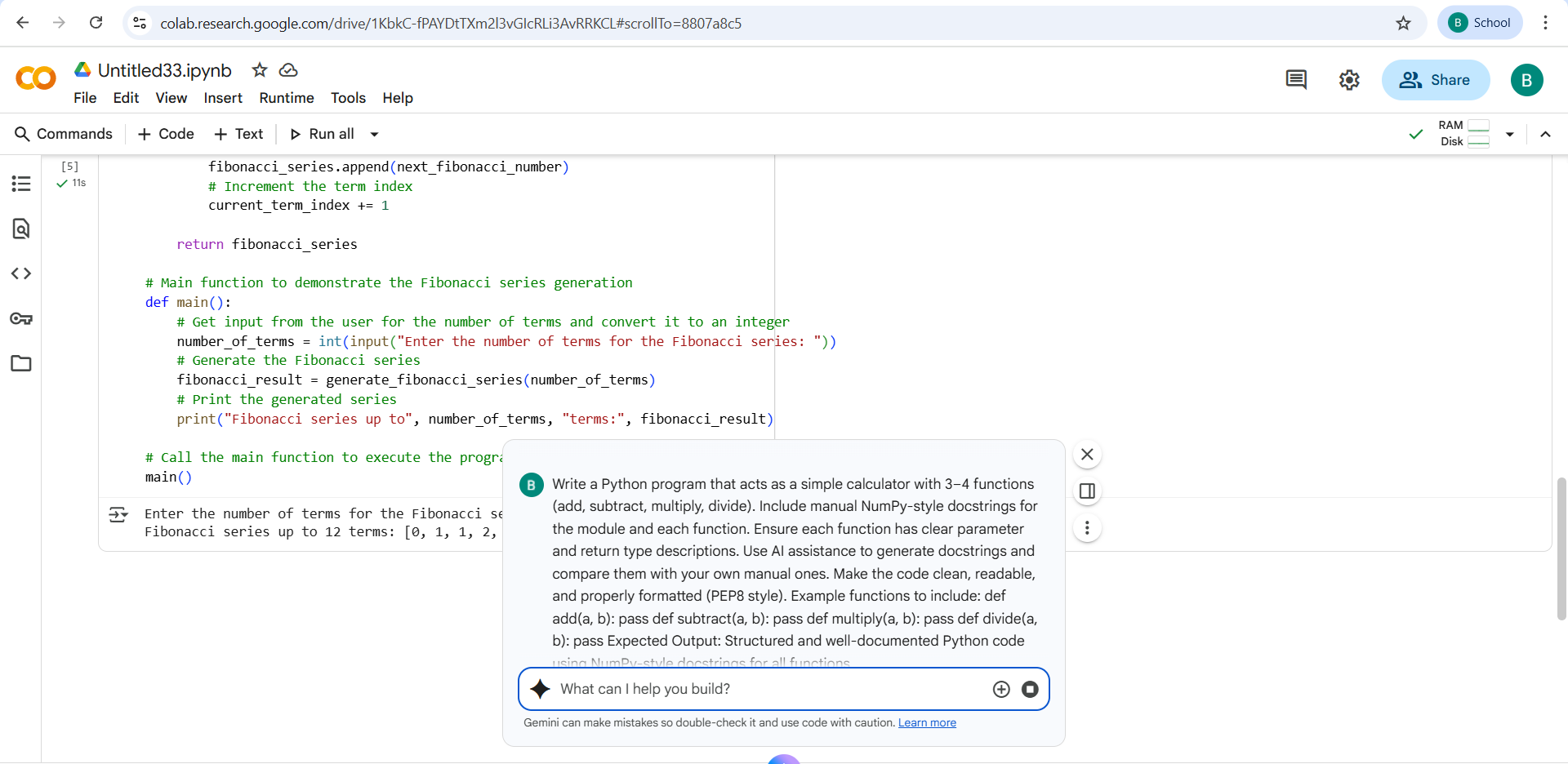
def multiply(a, b):

pass

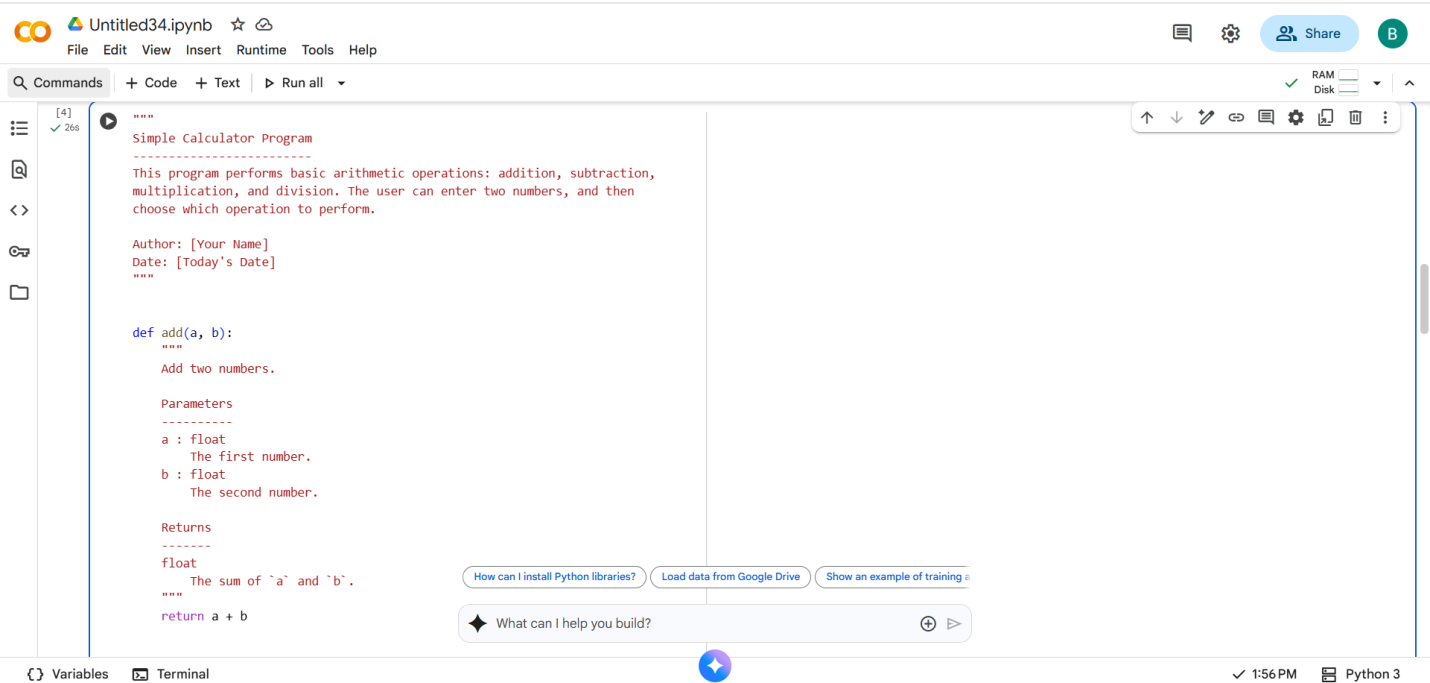
def divide(a, b):

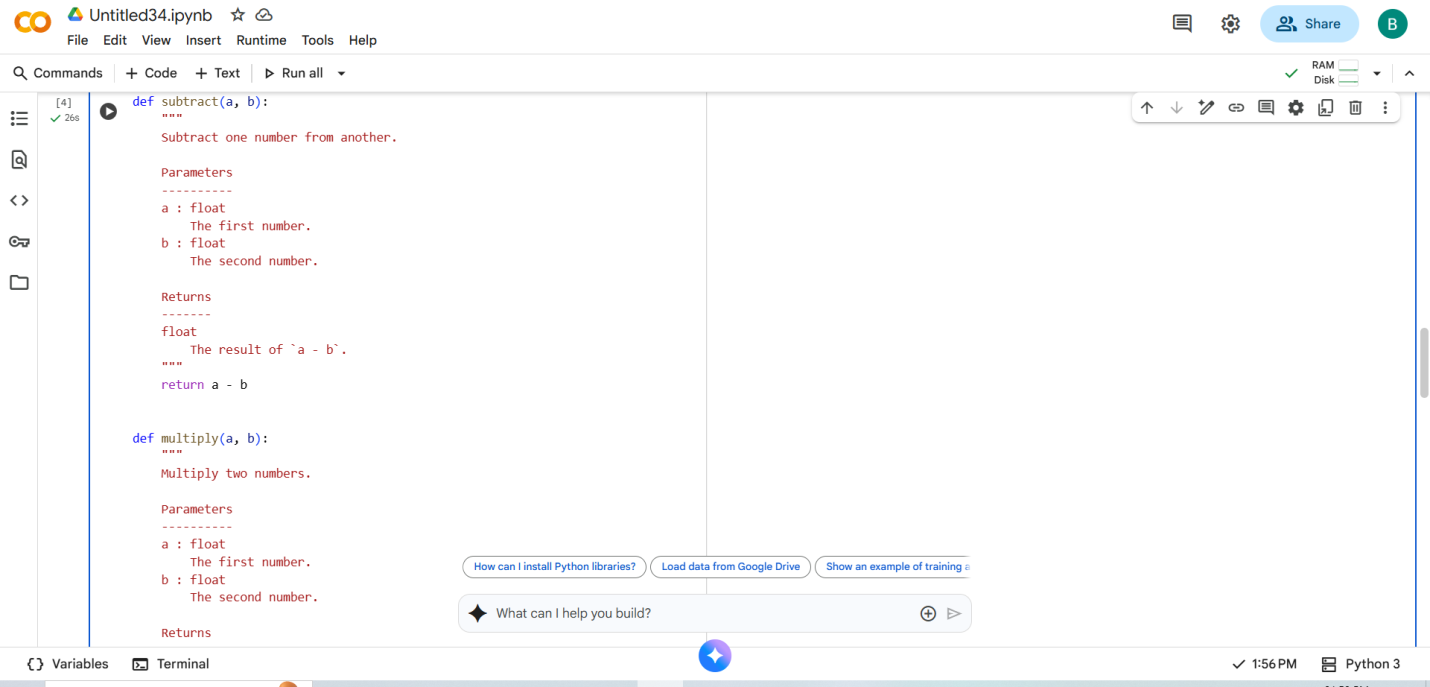
pass

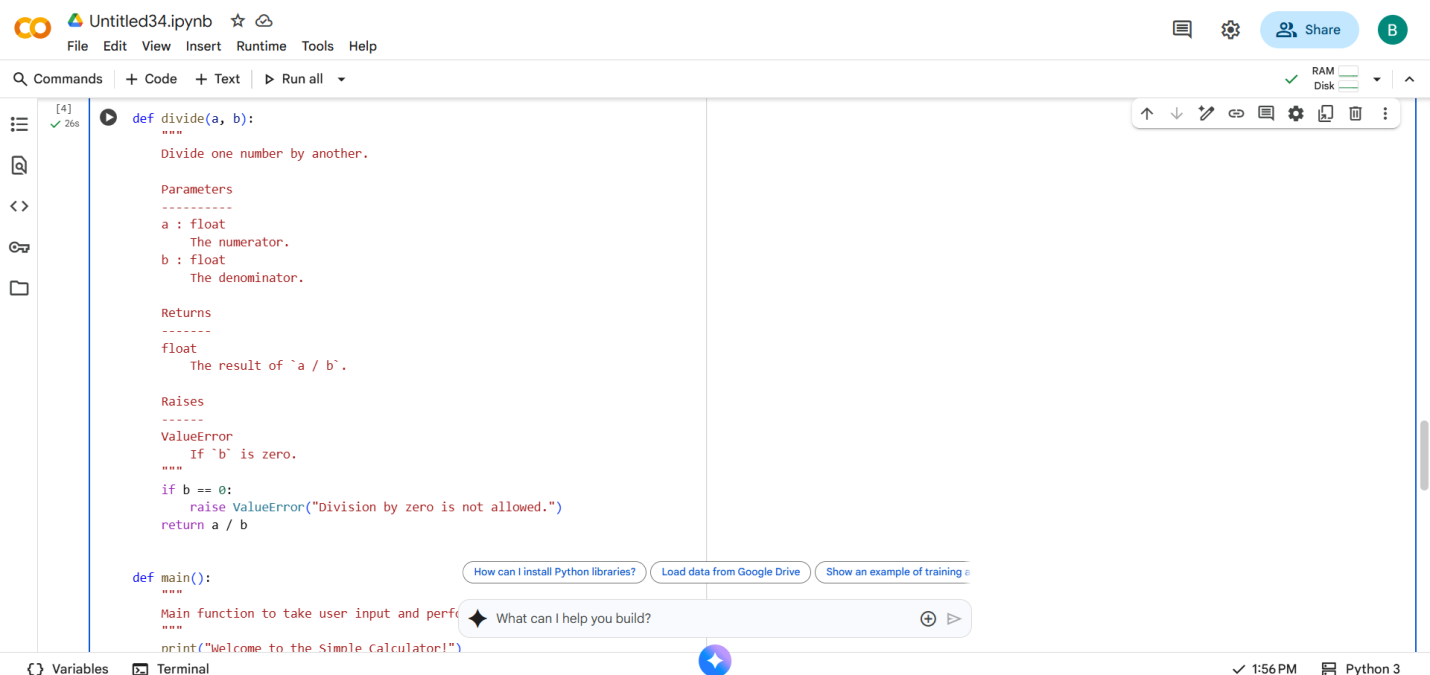
QUESTION:

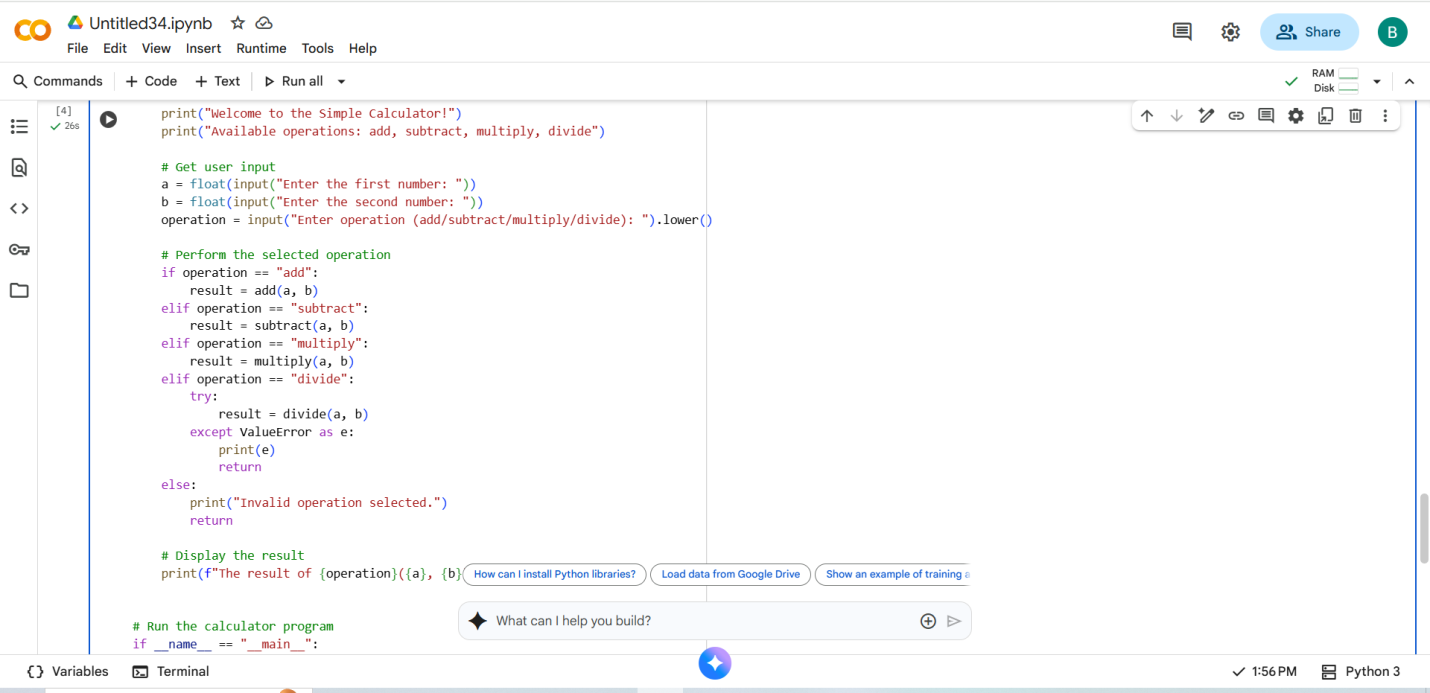


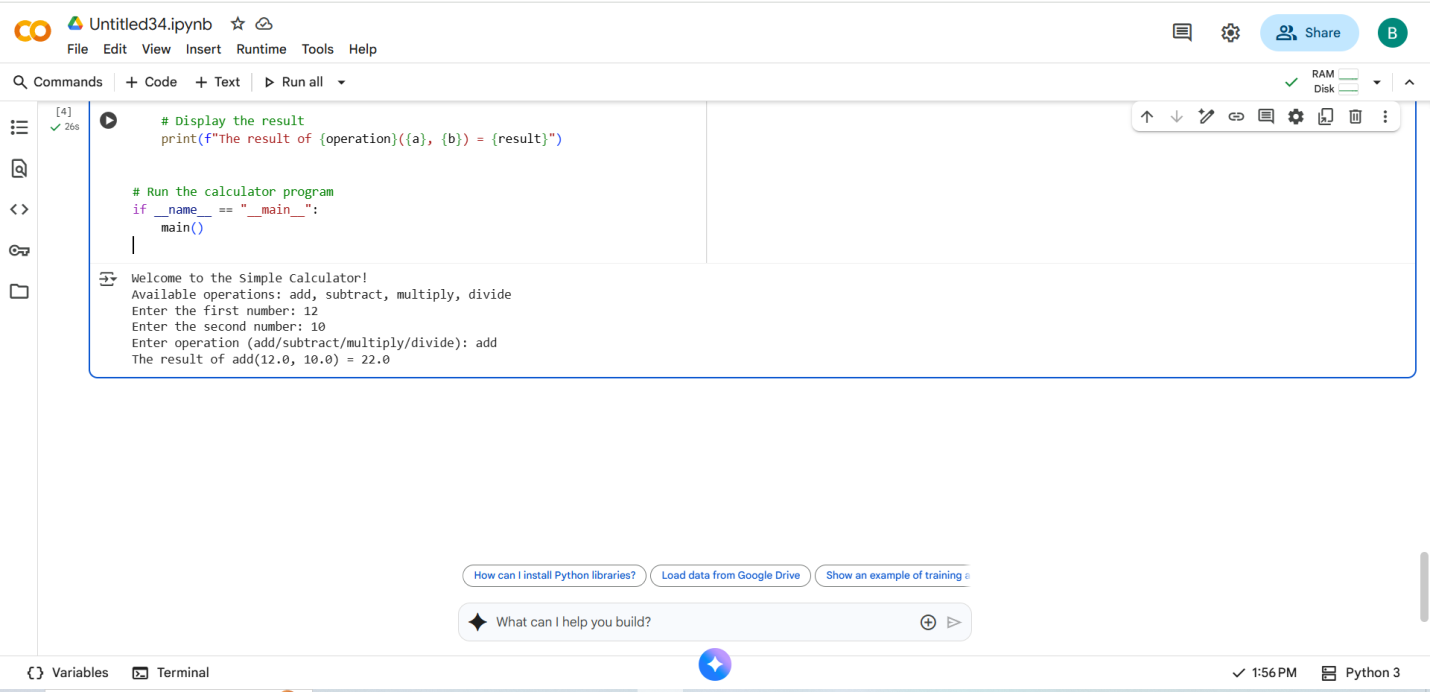
Code with output written by manual:



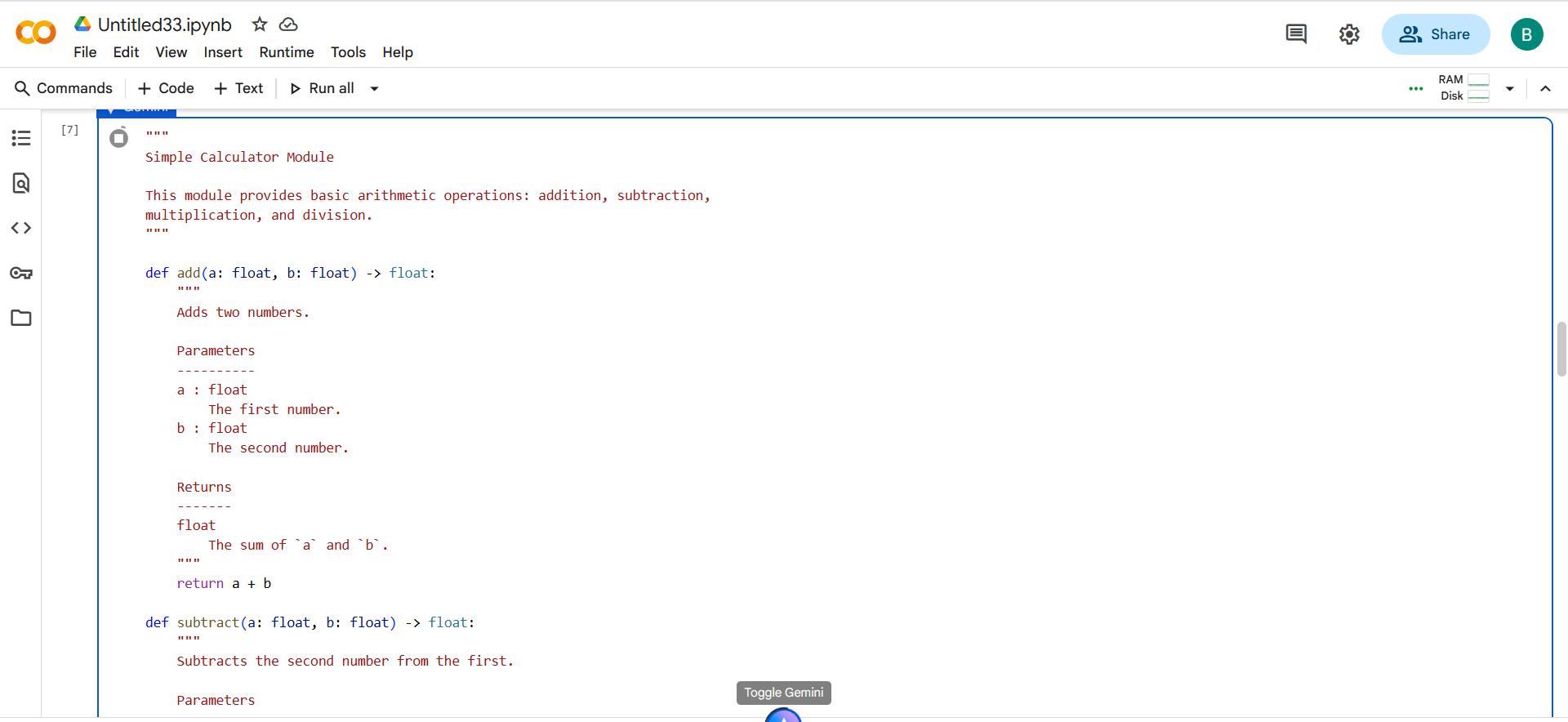


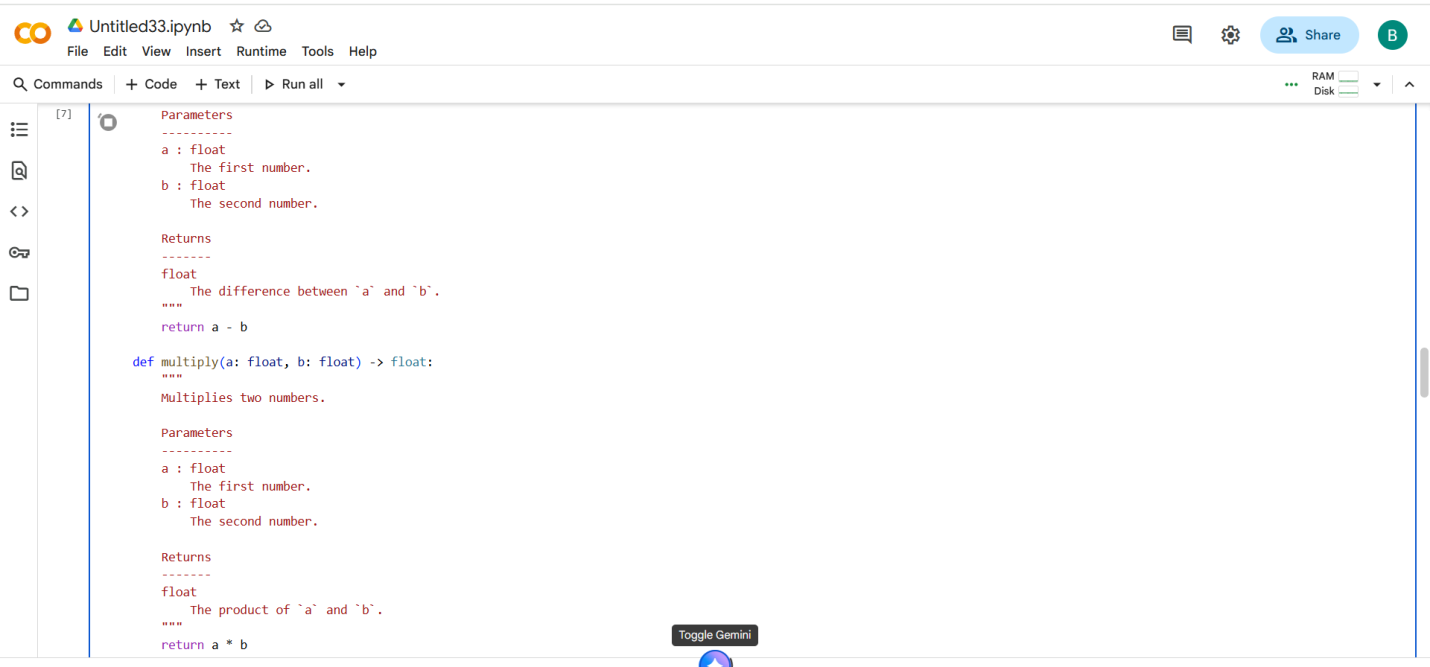


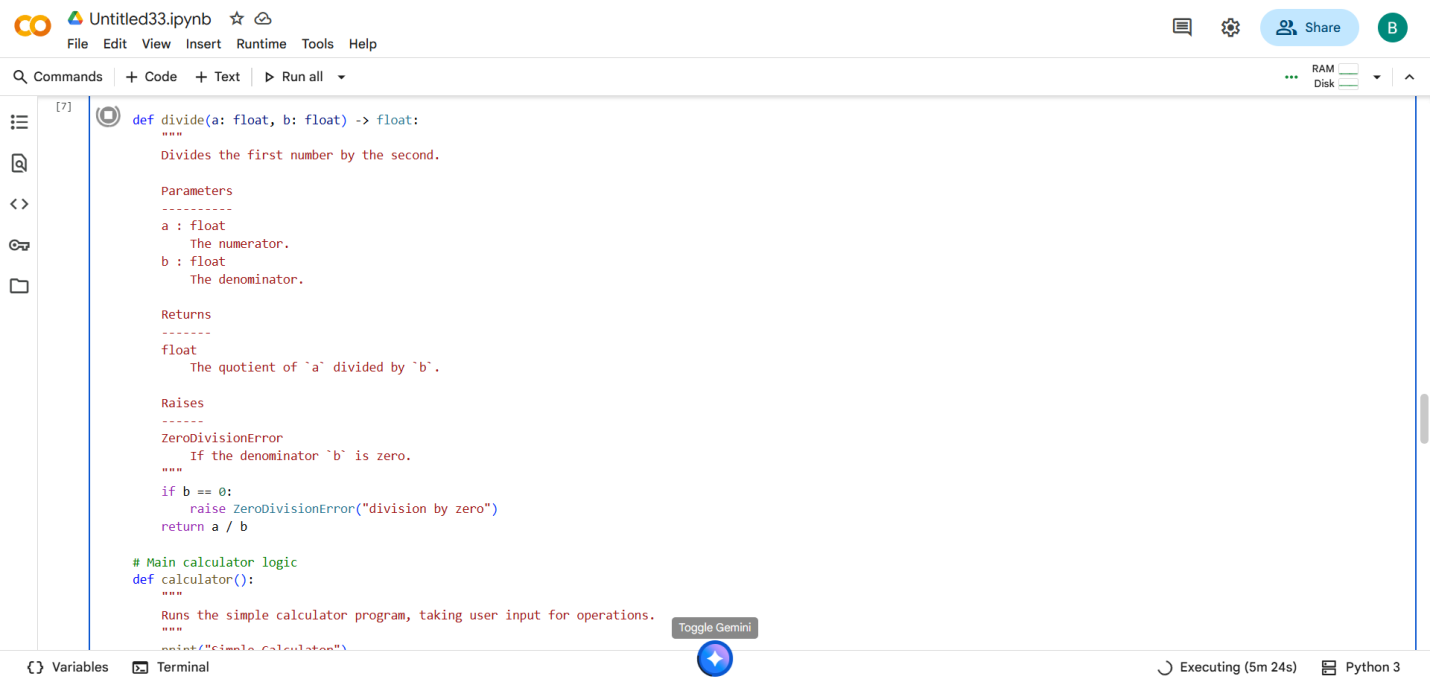


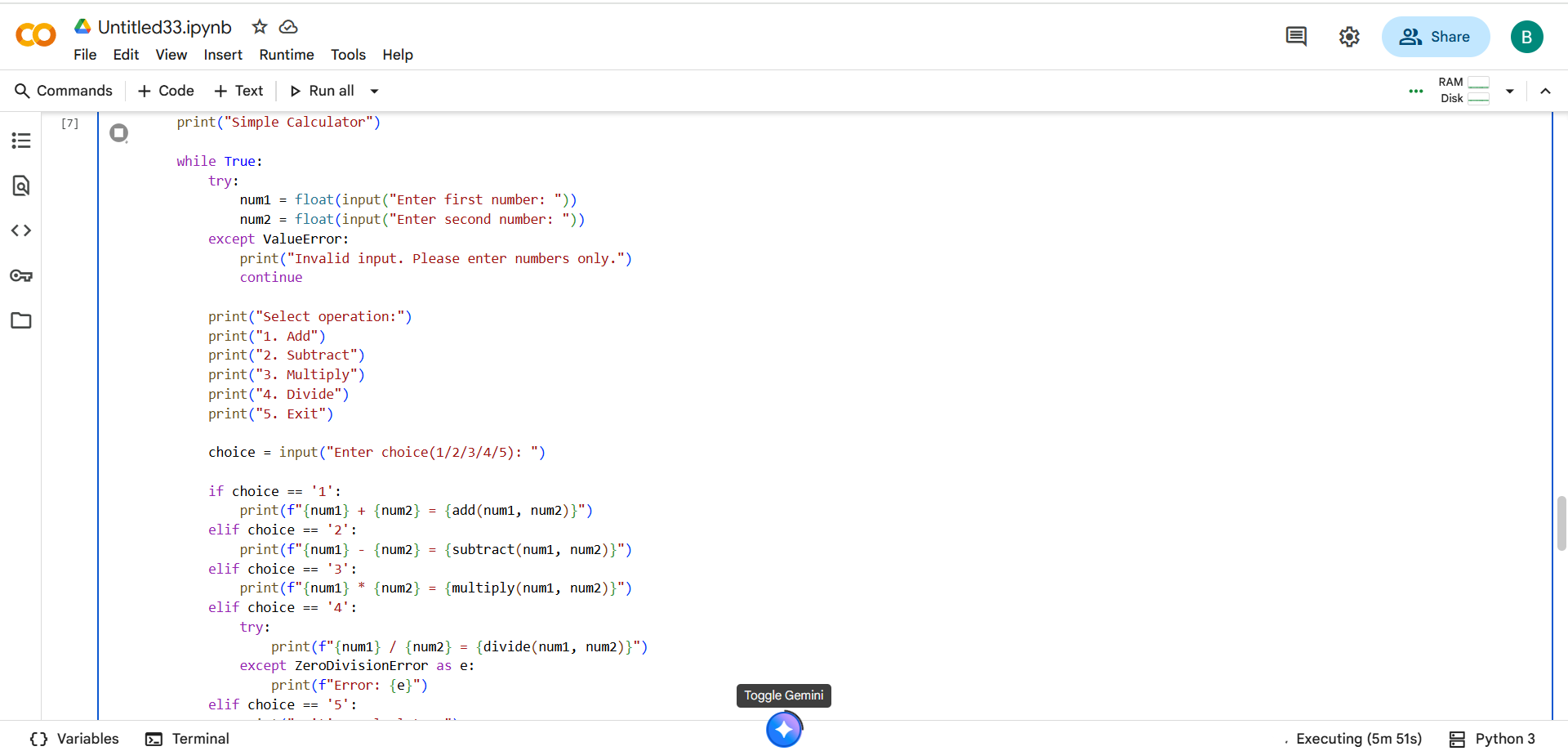


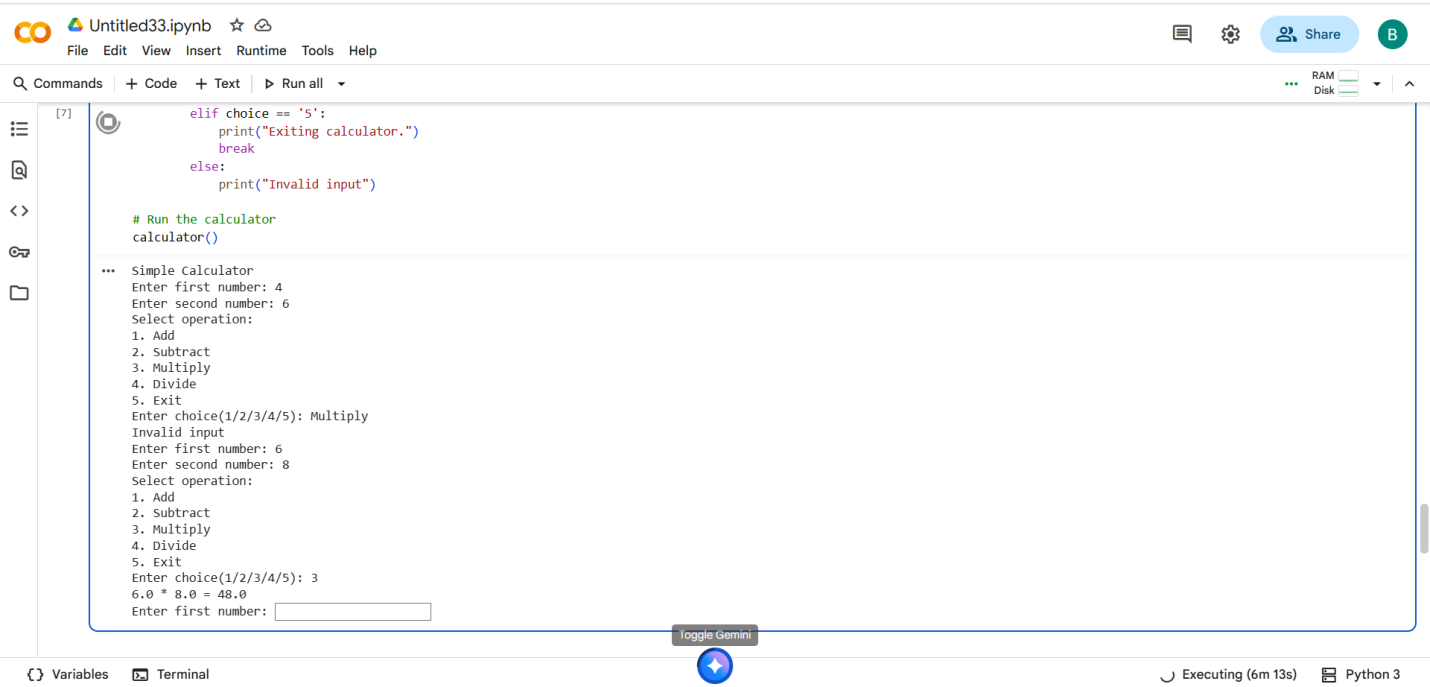
* Code with output using AI











* Comparing both AI and Manual code

-By comparing both codes Al generated best code than manual code and easily understandable code.

#EXPLANATION:

* **Module Docstring:** The code starts with a triple-quoted string at the beginning of the file. This is a module-level docstring that explains the overall purpose of the script – providing basic arithmetic operations.
* **add(a, b) function:** This function takes two floating-point numbers (a and b) as input and returns their sum. It includes a NumPy-style docstring explaining its parameters and what it returns.
* **subtract(a, b) function:** Similar to add, this function takes two floating-point numbers and returns their difference (a - b). It also has a NumPy-style docstring.
* **multiply(a, b) function:** This function takes two floating-point numbers and returns their product (a \* b), with a corresponding docstring.
* **divide(a, b) function:** This function takes two floating-point numbers and returns the result of dividing a by b. It includes a check to prevent division by zero, raising a ZeroDivisionError if the denominator is zero. Its docstring explains the parameters, return value, and the potential exception.
* **calculator() function:** This is the main function that runs the interactive calculator.
  + It enters an infinite while True loop to keep the calculator running until the user chooses to exit.
  + Inside the loop, it prompts the user to enter two numbers and handles potential ValueError if the input is not a valid number.
  + It then presents a menu of operations (Add, Subtract, Multiply, Divide, Exit) and prompts the user to select one.
  + Based on the user's choice, it calls the appropriate arithmetic function and prints the result.
  + It includes specific error handling for the divide function to catch ZeroDivisionError.
  + If the user enters '5', the loop breaks, and the program exits.
  + If the user enters an invalid choice, it prints an "Invalid input" message.
* **calculator() call:** The last line calculator() calls the main calculator function to start the program.

---------------THANK YOU-----------